Assignment #4

CSCI 581, Spring 2022

Jayaa Emekar

Predicting house prices in California

In this exercise, you will be working with the California housing dataset (regression) available from Scikit-learn. The original dataset is from *StatLib* (see http://lib.stat.cmu.edu/datasets/).

This dataset was derived from the 1990 U.S. census, using one row per census *block group*. A block group is the smallest geographical unit for which the U.S. Census Bureau publishes sample data (a block group typically has a population of 600 to 3,000 people).

A *household* is a group of people residing within a home. Since the average number of rooms and bedrooms in this dataset are provided per household, these columns may take surpinsingly large values for block groups with few households and many empty houses, such as vacation resorts.

Data

The dataset contains 20,640 observations on 9 numeric attributes. The target attribute is `MedHouseVal` , the median house value for California districts, expressed in hundreds of thousands of dollars ($100,000). The other attributes are:

- `MedInc` : median income in block group
- `HouseAge` : median house age in block group
- `AveRooms` : average number of rooms per household
- `AveBedrms` : average number of bedrooms per household
- `Population` : block group population
- `AveOccup` : average number of household members
- `Latitude` : block group latitude
- `Longitude` : block group longitude

Required components of your submission

Your *Google Colab* Jupyter notebook must include:

1. all pertinent *exploratory data analysis* (EDA) code, visualizations, and justifications;

    a) Confirm that there is no missing data.

b) Generate a histogram view of the all the attributes in the dataset using the `hist()` method in `pandas`.

c) Use the `Latitude` and `Longitude` attributes to visualize the locations associated with high-valued houses. Generate a scatter plot where the x- and y-axis would be the latitude and longitude, and the circle size and color would be linked with the house value in the district. What does the plot tell you?

d) Use the correlation matrix approach for feature selection and select a subset of the 8 predicting attributes to use for model building.

2. use of the `LinearRegression` estimator to build a model that will predict the median house value given the predicting attributes you selected

   a) Use the default values for a (multiple) linear regressor just like we did in class.

   b) Consider the use of a (multiple) polynomial regressor to improve the performance of your model, if applicable.

3. all pertinent model diagnostics, including metrics and visualizations; and

4. your final model equation.

Be sure to check out or review the *Assignments/Projects* section of our *Blackboard* course page for details regarding expectations, requirements, and the *Jupyter Notebook Rubric* that will be used to evaluate Jupyter notebook submissions.

Solution

Imports Required

In [ ]:
```
from sklearn.datasets import fetch_california_housing
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

Load the “California housing dataset”. This dataset can be fetched from internet using scikit-learn.

In [ ]:
```
#Load the “California housing dataset”. This dataset can be fetched from internet usir
california_housing = fetch_california_housing(as_frame=True)
#Look at the available description
print(california_housing.DESCR)
```

```
.. _california_housing_dataset:

California Housing dataset
--------------------------

**Data Set Characteristics:**

    :Number of Instances: 20640

    :Number of Attributes: 8 numeric, predictive attributes and the target

    :Attribute Information:
        - MedInc        median income in block group
        - HouseAge      median house age in block group
        - AveRooms      average number of rooms per household
        - AveBedrms     average number of bedrooms per household
        - Population    block group population
        - AveOccup      average number of household members
        - Latitude      block group latitude
        - Longitude     block group longitude

    :Missing Attribute Values: None

This dataset was obtained from the StatLib repository.
https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html

The target variable is the median house value for California districts,
expressed in hundreds of thousands of dollars ($100,000).

This dataset was derived from the 1990 U.S. census, using one row per census
block group. A block group is the smallest geographical unit for which the U.S.
Census Bureau publishes sample data (a block group typically has a population
of 600 to 3,000 people).

An household is a group of people residing within a home. Since the average
number of rooms and bedrooms in this dataset are provided per household, these
columns may take surpinsingly large values for block groups with few households
and many empty houses, such as vacation resorts.

It can be downloaded/loaded using the
:func:`sklearn.datasets.fetch_california_housing` function.

.. topic:: References

    - Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions,
      Statistics and Probability Letters, 33 (1997) 291-297
```

In [ ]:

```python
#Let’s have an overview of the entire dataset.
california_housing.frame.head()
```

Out[ ]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHou
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	

As written in the description, the dataset contains aggregated data regarding each district in California. Let's have a close look at the features that can be used by a predictive model.

In [ ]:
```
#Let's have a close look at the features that can be used by a predictive model
california_housing.data.head()
```

Out[ ]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25

This dataset, we have information regarding the demography (income, population, house occupancy) in the districts, the location of the districts (latitude, longitude), and general information regarding the house in the districts (number of rooms, number of bedrooms, age of the house). Since these statistics are at the granularity of the district, they corresponds to averages or medians.

Now, let's have a look to the target to be predicted.

In [ ]:
```
# Display the values of the target attribute
california_housing.target.head()
```

Out[ ]:
```
0    4.526
1    3.585
2    3.521
3    3.413
4    3.422
Name: MedHouseVal, dtype: float64
```

The target contains the median of the house value for each district. Therefore, this problem is a regression problem.

We can now check more into details the data types and if the dataset contains any missing value.

```
#check for missing or null values
```

In [ ]:
```
california_housing.frame.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   MedInc       20640 non-null  float64
 1   HouseAge     20640 non-null  float64
 2   AveRooms     20640 non-null  float64
 3   AveBedrms    20640 non-null  float64
 4   Population   20640 non-null  float64
 5   AveOccup     20640 non-null  float64
 6   Latitude     20640 non-null  float64
 7   Longitude    20640 non-null  float64
 8   MedHouseVal  20640 non-null  float64
dtypes: float64(9)
memory usage: 1.4 MB
```

We can see that:

1. the dataset contains 20,640 samples and 8 features;
2. all features are numerical features encoded as floating number;
3. there is no missing values.

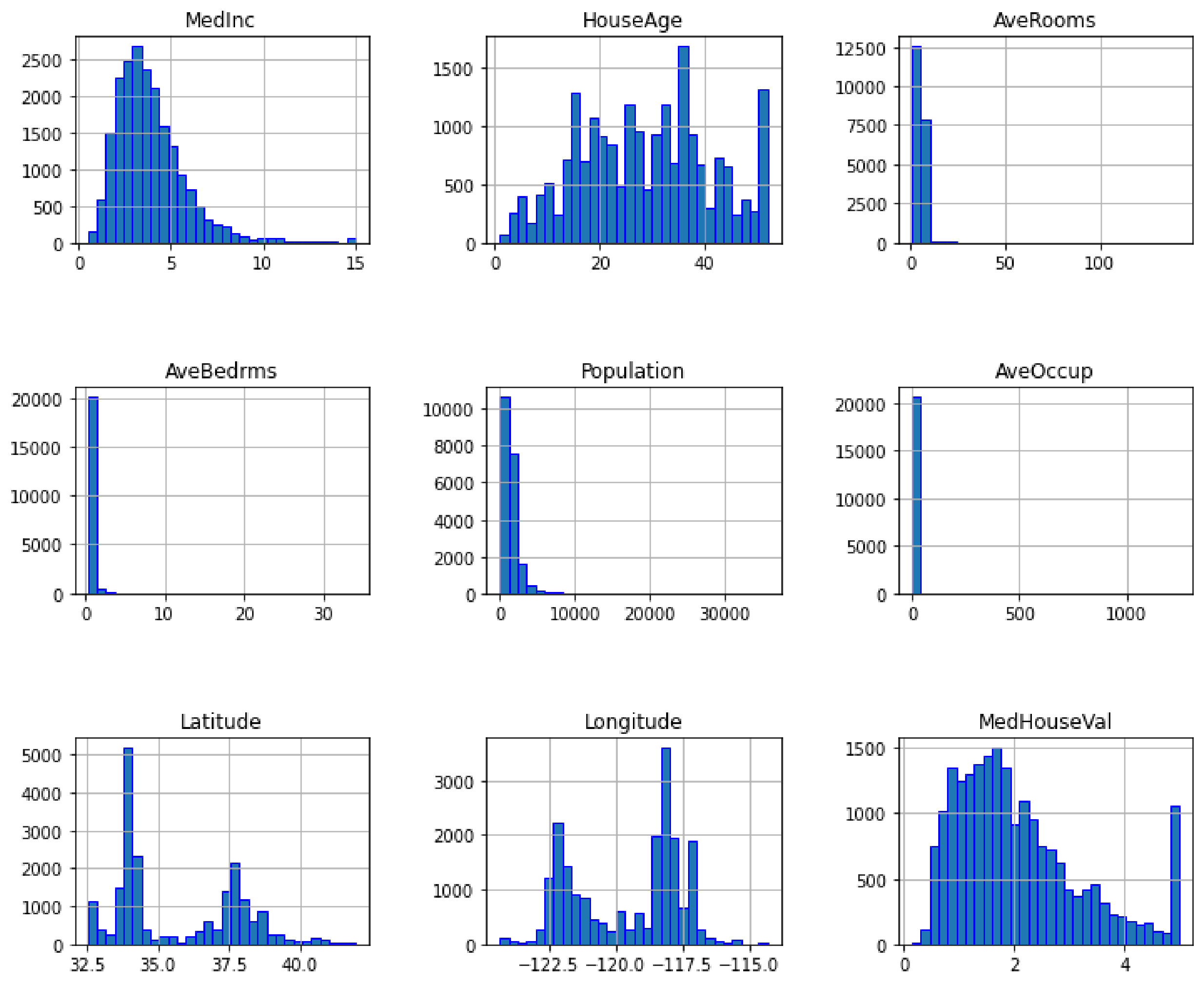
In [ ]:
```
# Search for any possible missing values
california_housing_df = pd.DataFrame(california_housing.data, columns=california_housi
print(california_housing_df.isnull().sum())
```

```
MedInc        0
HouseAge      0
AveRooms      0
AveBedrms     0
Population    0
AveOccup      0
Latitude      0
Longitude     0
dtype: int64
```

This confirms that there is no missing values in the California Housing Data

Let's have a quick look at the distribution of these features by plotting their histograms

In [ ]:
```
#Plot histogram for California Housing dataset
california_housing.frame.hist(figsize=(12, 10), bins=30, edgecolor="blue")
plt.subplots_adjust(hspace=0.7, wspace=0.4)
```



From above graph we can conclude below:

1. Focus on features for which their distributions would be more or less expected.
2. The median income is a distribution has a long tail. It means that the salary of people is more or less normally distributed but there is few people getting a high salary.
3. Regarding the average house age, the distribution is more or less uniform.
4. For high-valued houses: all houses with a price above 5 are given the value 5.
5. Focusing on the other features like average rooms, average bedrooms, average occupation, and population, the range of the data is large with unnoticeable bin for the largest values. It means that there are very high and few values.

In [ ]:
```
# Specificity looking at the statistics for these features:
features_of_interest = ["AveRooms", "AveBedrms", "AveOccup", "Population"]
california_housing.frame[features_of_interest].describe()
```

Out[ ]:

	AveRooms	AveBedrms	AveOccup	Population
count	20640.000000	20640.000000	20640.000000	20640.000000
mean	5.429000	1.096675	3.070655	1425.476744
std	2.474173	0.473911	10.386050	1132.462122
min	0.846154	0.333333	0.692308	3.000000
25%	4.440716	1.006079	2.429741	787.000000
50%	5.229129	1.048780	2.818116	1166.000000
75%	6.052381	1.099526	3.282261	1725.000000
max	141.909091	34.066667	1243.333333	35682.000000

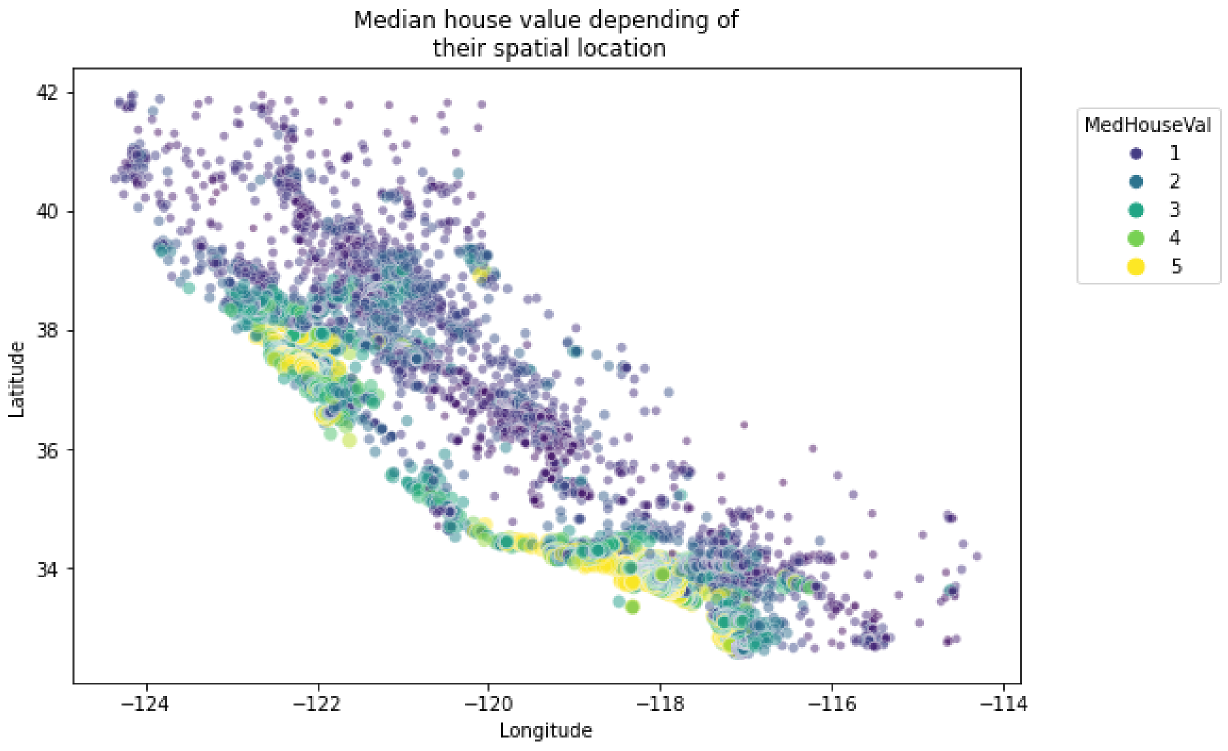
For each of these features, comparing the max and 75% values, we can see a huge difference. It confirms the intuitions that there are a couple of extreme values.

We discarded the longitude and latitude that carry geographical information. The combination of this feature could help us to decide if there are locations associated with high-valued houses.

Lets make a scatter plot where the x- and y-axis would be the latitude and longitude and the circle size and color would be linked with the house value in the district.

In [ ]:

```
#Scatter plot median house value depending upon their  latitude and longitude
plt.figure(figsize=(9, 6))
sns.scatterplot(data=california_housing.frame, x="Longitude", y="Latitude",
                size="MedHouseVal", hue="MedHouseVal",
                palette="viridis", alpha=0.5)
plt.legend(title="MedHouseVal",bbox_to_anchor=(1.05, 0.95),loc="upper left")
plt.title("Median house value depending of\n their spatial location")
plt.show()
```

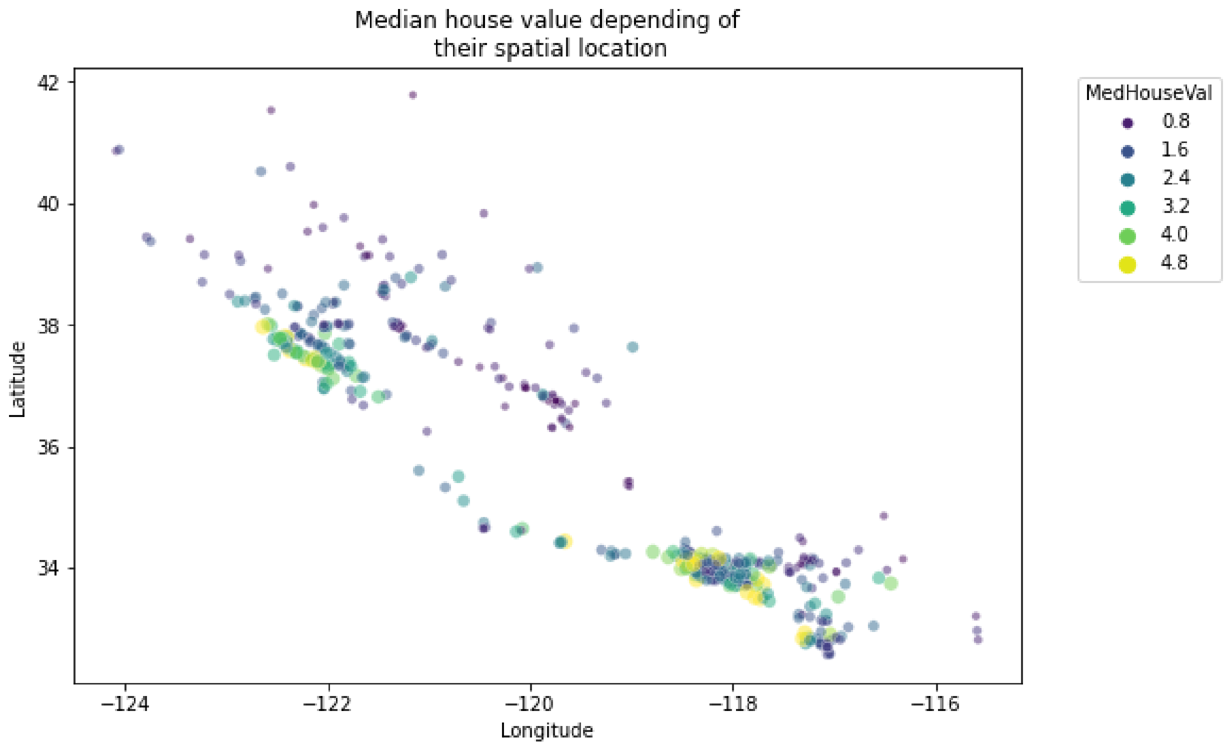


Conclusion from graph:

1. For state of California, it is interesting to notice that all datapoints show a graphical representation of this state.
2. We note that the high-valued houses will be located on the coast, where the big cities from California are located: San Diego, Los Angeles, San Jose, or San Francisco.

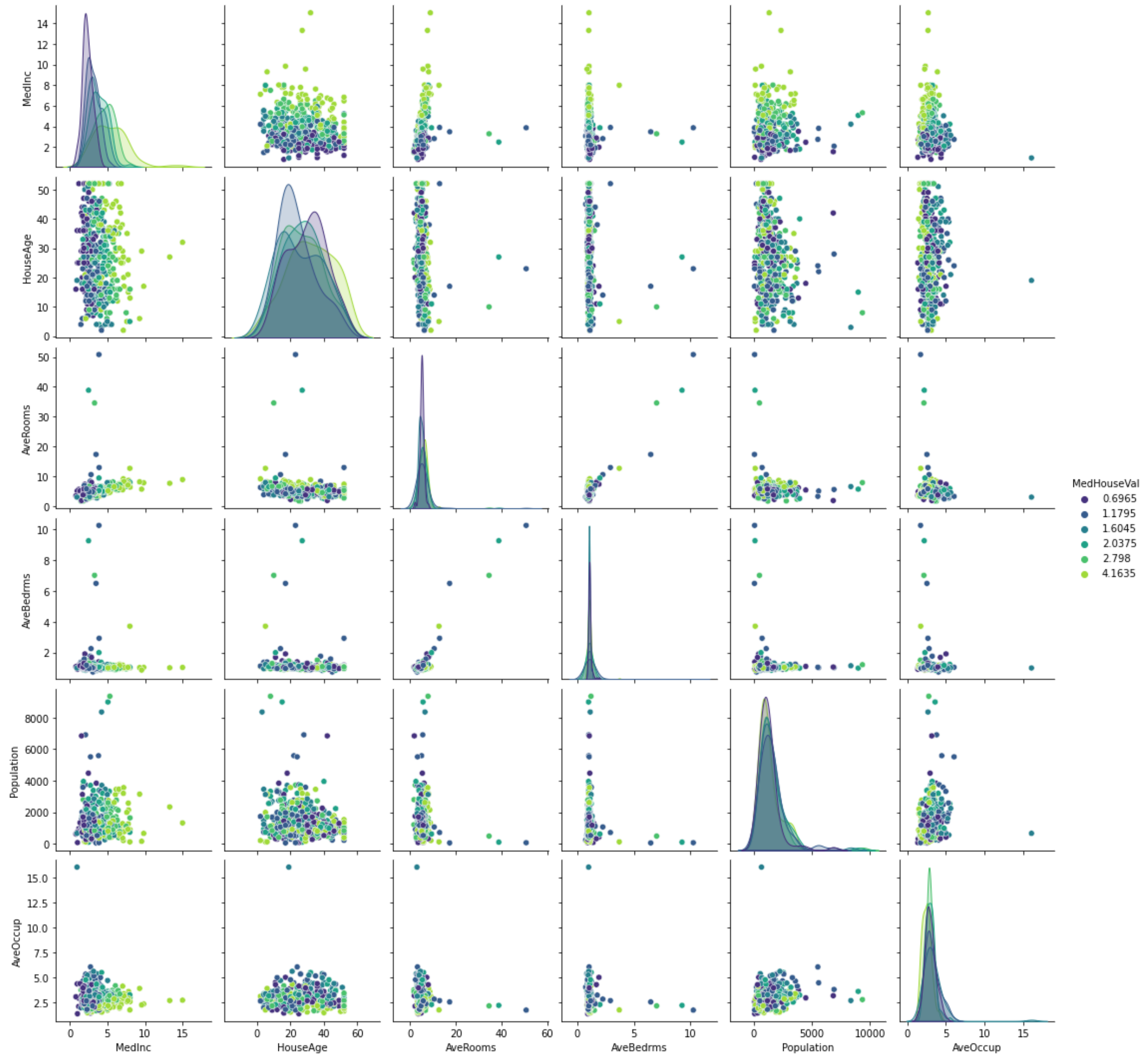
Lets do a random subsampling to have less data points to plot but that could still allow us to see these specificities.

In [ ]:
```
# Lets do a random subsampling to have less data points to plot
# but that could still allow us to see these specificities.
rng = np.random.RandomState(0)
indices = rng.choice(np.arange(california_housing.frame.shape[0]), size=500,
                     replace=False)
plt.figure(figsize=(9, 6))
sns.scatterplot(data=california_housing.frame.iloc[indices],
                x="Longitude", y="Latitude",
                size="MedHouseVal", hue="MedHouseVal",
                palette="viridis", alpha=0.5)
plt.legend(title="MedHouseVal", bbox_to_anchor=(1.05, 1),
           loc="upper left")
plt.title("Median house value depending of\n their spatial location")
plt.show()
```



For the final analysis by making a pair plot of all features and the target but dropping the longitude and latitude. We will quantize the target such that we can create proper histogram.

```
In [ ]: # Drop the unwanted columns
        columns_drop = ["Longitude", "Latitude"]

        subset = california_housing.frame.iloc[indices].drop(columns=columns_drop)
        # Quantize the target and keep the midpoint for each interval
        subset["MedHouseVal"] = pd.qcut(subset["MedHouseVal"], 6, retbins=False)
        subset["MedHouseVal"] = subset["MedHouseVal"].apply(lambda x: x.mid)
        sns.pairplot(data=subset, hue="MedHouseVal", palette="viridis")
        plt.show()
```



Conclusion from plot:

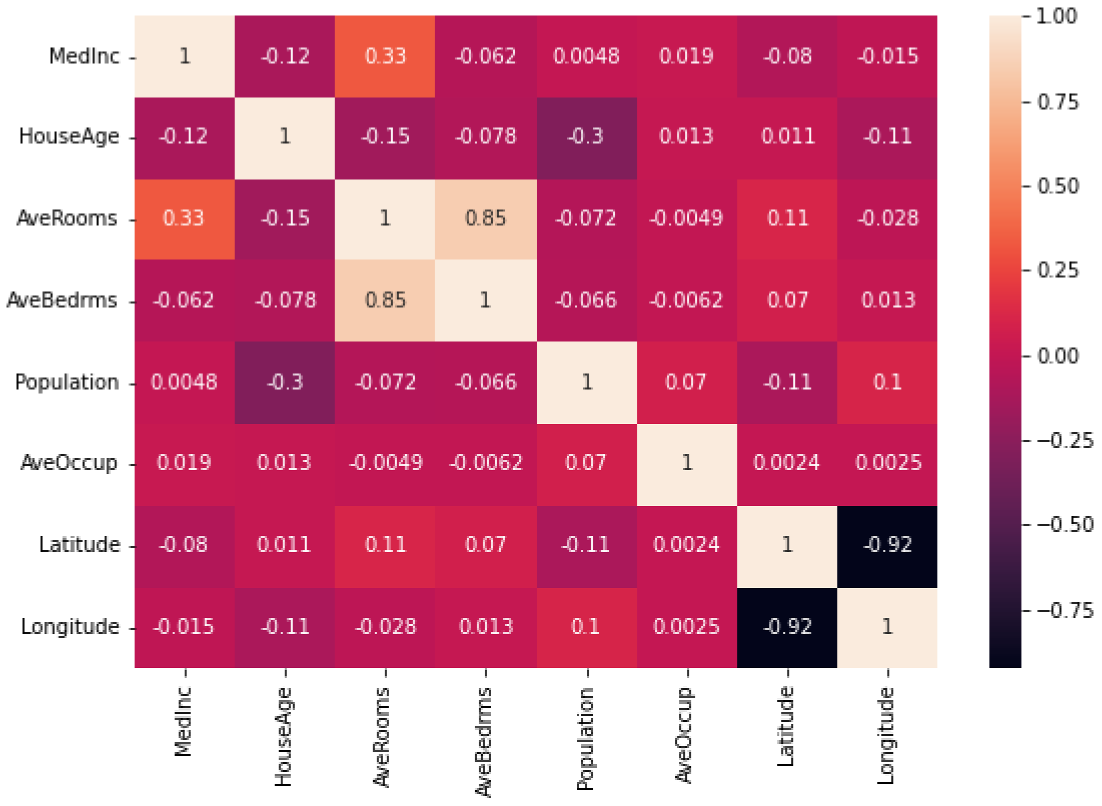
1. Its always complicated to interpret a pairplot since there is a lot of data, here we can get a couple of intuitions. We can confirm that some features have extreme values.
2. We can see that the median income is helpful to distinguish high-valued from low-valued houses.
3. Thus, creating a predictive model, we could expect the longitude, latitude, and the median income to be useful features to help at predicting the median house values.

Feature Selection

In [ ]:
```
# Compute the pairwise correlation of features/attributes/columns
corr = california_housing_df.corr()
# Print the correlation matrix
print(corr)
```

```
              MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  \
MedInc      1.000000 -0.119034  0.326895  -0.062040    0.004834  0.018766
HouseAge   -0.119034  1.000000 -0.153277  -0.077747   -0.296244  0.013191
AveRooms    0.326895 -0.153277  1.000000   0.847621   -0.072213 -0.004852
AveBedrms  -0.062040 -0.077747  0.847621   1.000000   -0.066197 -0.006181
Population  0.004834 -0.296244 -0.072213  -0.066197    1.000000  0.069863
AveOccup    0.018766  0.013191 -0.004852  -0.006181    0.069863  1.000000
Latitude   -0.079809  0.011173  0.106389   0.069721   -0.108785  0.002366
Longitude  -0.015176 -0.108197 -0.027540   0.013344    0.099773  0.002476

            Latitude  Longitude
MedInc     -0.079809  -0.015176
HouseAge    0.011173  -0.108197
AveRooms    0.106389  -0.027540
AveBedrms   0.069721   0.013344
Population -0.108785   0.099773
AveOccup    0.002366   0.002476
Latitude    1.000000  -0.924664
Longitude  -0.924664   1.000000
```

In [ ]:
```python
# Plot the correlation matrix
plt.figure(figsize=(9, 6))
sns.heatmap(corr, annot=True)
plt.show()
```



In [ ]:
```python
#---get the top 3 features that has the highest correlation---
california_housing_df = pd.DataFrame(california_housing.data, columns=california_housi
california_housing_df['MedHouseVal'] = california_housing.target
print(california_housing_df.corr().abs().nlargest(3, 'MedHouseVal').index) # index sho
print(california_housing_df.corr().abs().nlargest(3, 'MedHouseVal').to_numpy()[:,8])
```

```
Index(['MedHouseVal', 'MedInc', 'AveRooms'], dtype='object')
[1.         0.68807521 0.15194829]
```

Note that from the dataset description

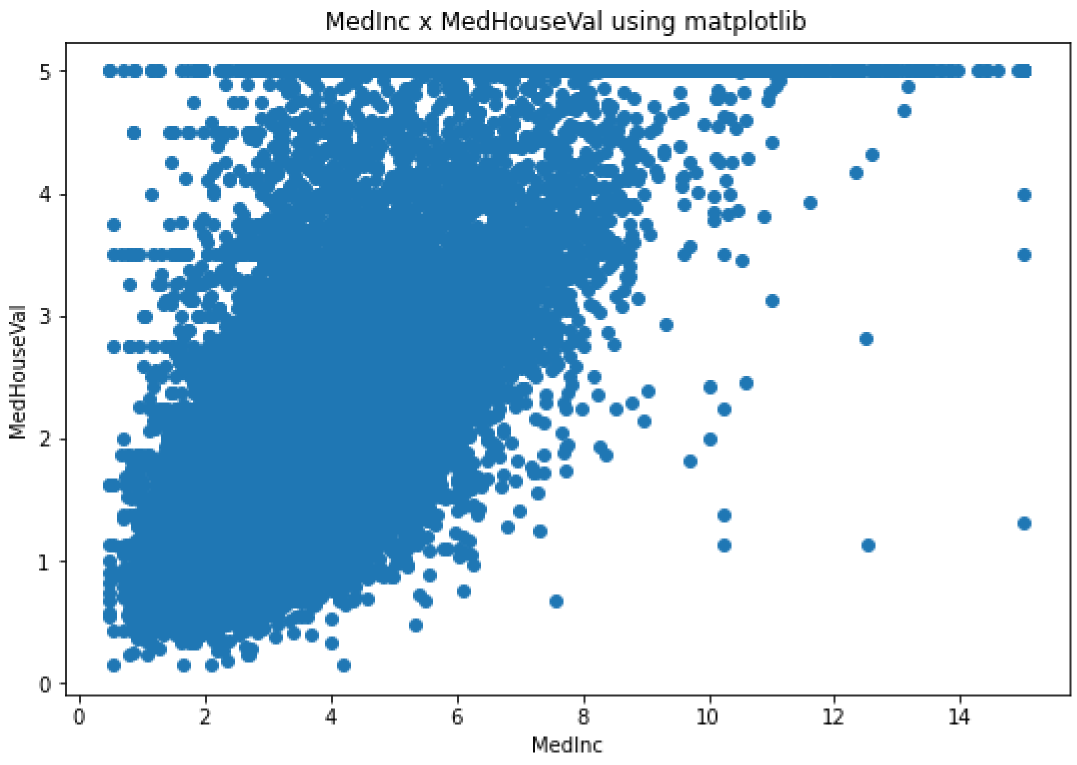
- `MedInc` : median income in block group
- `AveRooms` : average number of rooms per household
- `MedHouseVal` : The target attribute is `MedHouseVal` , the median house value for California districts, expressed in hundreds of thousands of dollars ($100,000).

Multiple Linear Regression

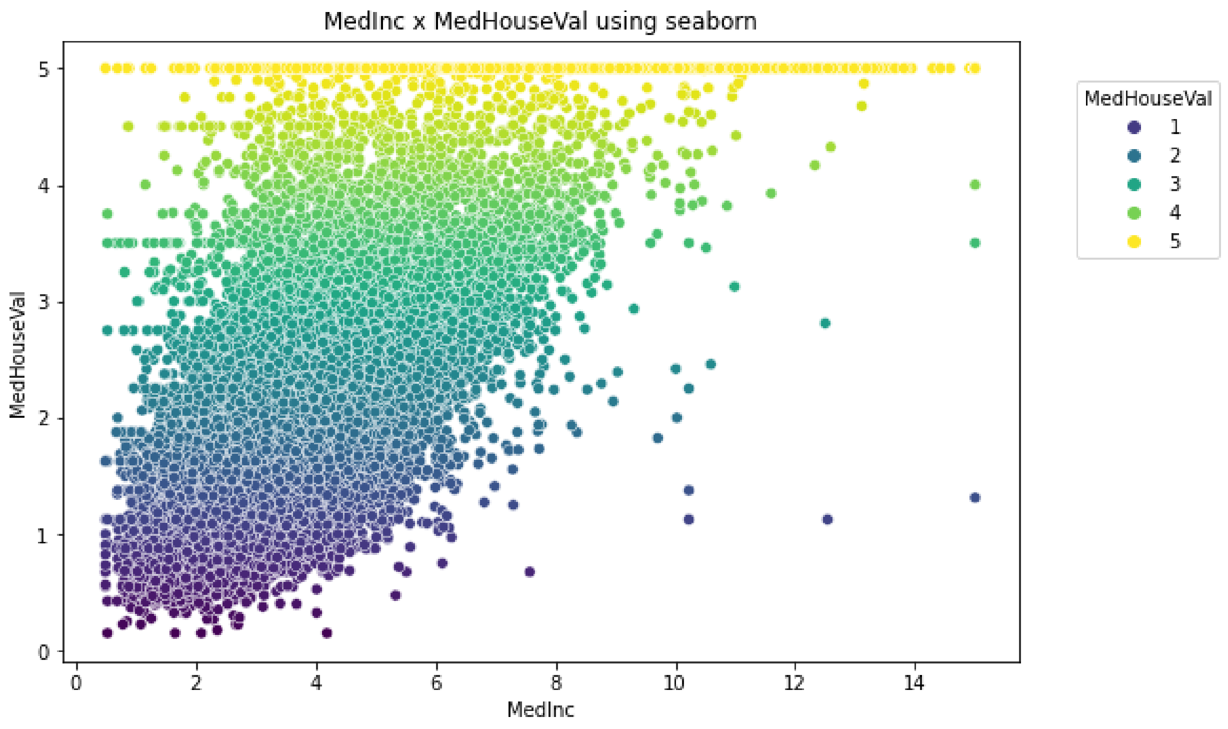
You perform a simple linear regression using a single feature and a label. here, we want to train your model using more than one independent variable and a label. This is known as multiple regression. In multiple regression, two or more independent variables are used to predict the value of a dependent variable (label).

Plot against 'MedInc' vs 'MedHouseVal'

In [ ]:
```
plt.figure(figsize=(9, 6))
plt.scatter(california_housing_df['MedInc'], california_housing_df['MedHouseVal'], mar
plt.title('MedInc x MedHouseVal using matplotlib')
# Median income in block group
plt.xlabel('MedInc')
#  The median house value for California districts (in $1,00000s)
plt.ylabel('MedHouseVal')
plt.show()
```

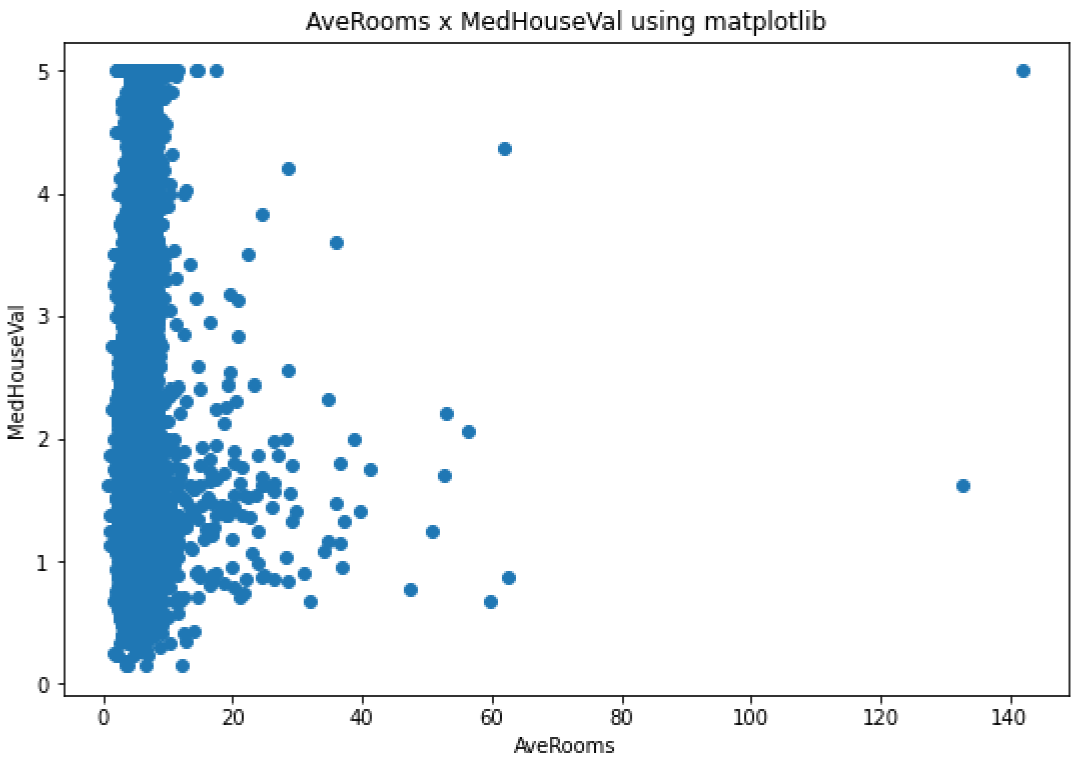


In [ ]:
```
plt.figure(figsize=(9, 6))
sns.scatterplot(
        data=california_housing_df,
        x=california_housing_df['MedInc'],
        y=california_housing_df['MedHouseVal'],
        hue=california_housing_df['MedHouseVal'],
        palette="viridis",
        legend=True
        )
plt.title('MedInc x MedHouseVal using seaborn')
plt.legend(title="MedHouseVal",bbox_to_anchor=(1.05, 0.95),loc="upper left")
# Median income in block group
plt.xlabel('MedInc')
#  The median house value for California districts (in $1,00000s)
plt.ylabel('MedHouseVal')
plt.show()
```

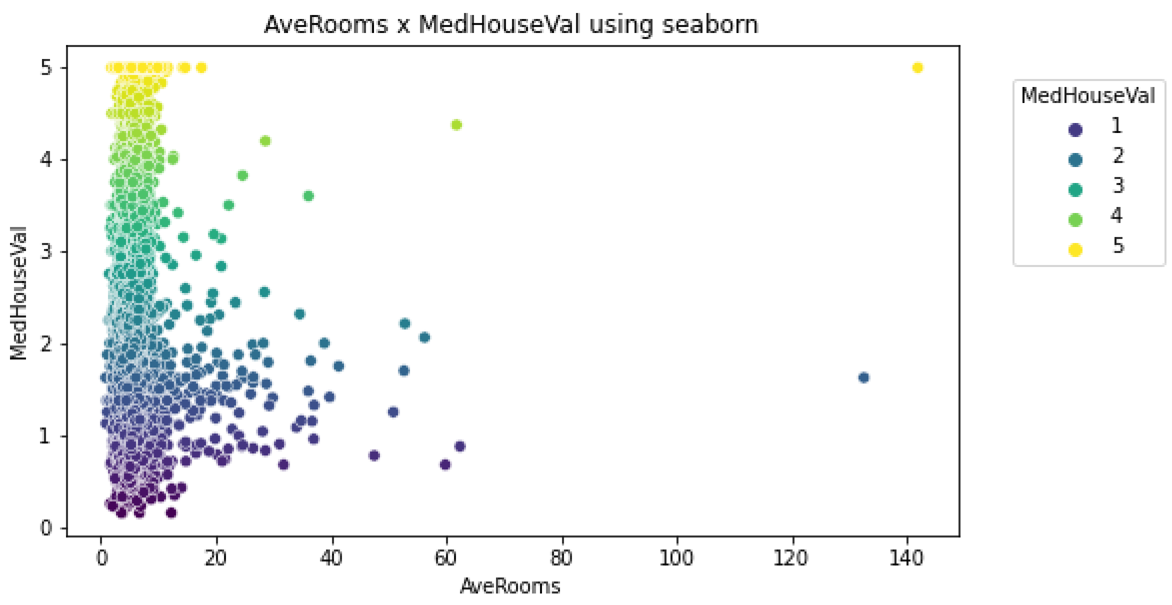


Plot against 'AveRooms' vs 'MedHouseVal'

In [ ]:
```
plt.figure(figsize=(9, 6))
plt.scatter(california_housing_df['AveRooms'], california_housing_df['MedHouseVal'], m
plt.title('AveRooms x MedHouseVal using matplotlib')
# average number of rooms per household
plt.xlabel('AveRooms')
#  The median house value for California districts (in $1,00000s)
plt.ylabel('MedHouseVal')
plt.show()
```



```
In [ ]: plt.figure(figsize=(8, 4.5))
        sns.scatterplot(
                data=california_housing_df,
                x=california_housing_df['AveRooms'],
                y=california_housing_df['MedHouseVal'],
                hue=california_housing_df['MedHouseVal'],
                palette="viridis",
                legend=True
                )
        plt.title('AveRooms x MedHouseVal using seaborn')
        plt.legend(title="MedHouseVal",bbox_to_anchor=(1.05, 0.95),loc="upper left")
        # average number of rooms per household
        plt.xlabel('AveRooms')
        #  The median house value for California districts (in $1,00000s)
        plt.ylabel('MedHouseVal')
        plt.show()
```



Training the Model

We can now train the model. First, create two DataFrames: x and Y. The x DataFrame will contain the combination of the MedInc and AveRooms features, while the Y DataFrame will contain the AveRooms label.

```
In [ ]: # Concatenate slices to create x values for training
        x = pd.DataFrame(np.c_[california_housing_df['MedInc'], california_housing_df['AveRoom
        columns = ['MedInc','AveRooms'])
        Y = california_housing_df['MedHouseVal']
```

We will split the dataset into 70 percent for training and 30 percent for testing

```
In [ ]: x_train, x_test, Y_train, Y_test = train_test_split(x, Y, test_size = 0.3,random_state
```

After the split, let's print out the shape of the training sets

```
In [ ]: print(x_train.shape)
        print(Y_train.shape)
```

```
(14448, 2)
(14448,)
```

This means that the x training set now has 14448 rows and 2 columns, while the Y training set (which contains the label) has 14448 rows and 1 column.

Let's also print out the testing set

```
In [ ]: print(x_test.shape)
        print(Y_test.shape)
```

```
(6192, 2)
(6192,)
```

the testing set has 6129 rows

We are now ready to begin the training.you can use the LinearRegression class to perform linear regression. In this case, we will use it to train our model

In [ ]:
```python
# Use linear regression
model = LinearRegression()
model.fit(x_train, Y_train)

# Get the parameters for this estimator
params = model.get_params()
print('Model parameters:', params)
```

```
Model parameters: {'copy_X': True, 'fit_intercept': True, 'n_jobs': None, 'normaliz
e': 'deprecated', 'positive': False}
```

Once the model is trained, we will use the testing set to perform some predictions

In [ ]:
```python
price_pred = model.predict(x_test)

print("Actual/observed price ",Y_test) # actual/observed price
print(type(Y_test))
print("Predicted price ",price_pred) # predicted price
print(type(price_pred))
```

```
Actual/observed price  14772    0.936
10105    1.536
20094    1.325
19261    1.479
14139    1.207
         ...
15291    1.688
1595     4.065
8755     3.788
1294     1.681
7010     1.632
Name: MedHouseVal, Length: 6192, dtype: float64
<class 'pandas.core.series.Series'>
Predicted price  [1.61331406 1.98497175 0.284646   ... 2.38633677 2.09999021 2.209647
99]
<class 'numpy.ndarray'>
```

Calculate R^2 to determine how well the model performs with the testing set.

To learn how well our model performed, we use the R-Squared method. The R-Squared method lets you know how close the test data fits the regression line. A value of 1.0 means a perfect fit. So, you aim for a value of R-Squared that is close to 1

In [ ]:
```python
print('R-squared: %.4f' % model.score(x_test,Y_test))
```

```
R-squared: 0.4924
```

Calculate the mean squared error (MSE) between actual/observed values and predicted values for price.

Generate a scatter plot to visualize the differences between these values.

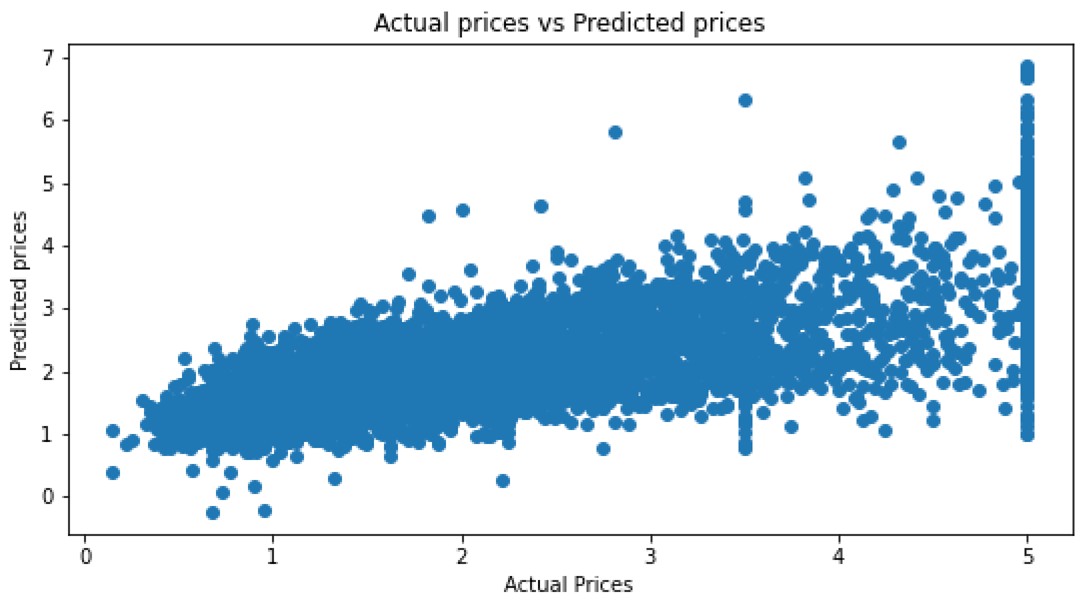
In [ ]:
```python
mse = mean_squared_error(Y_test, price_pred)
```

```
print('MSE =', mse)
plt.figure(figsize=(9, 4.5))
plt.scatter(Y_test, price_pred)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted prices")
plt.title("Actual prices vs Predicted prices")
```

```
MSE = 0.6919541480004132
```

Out[ ]:

```
Text(0.5, 1.0, 'Actual prices vs Predicted prices')
```



The formula for multiple regression is as follows: Y = β0 + β1x1 +β2x2

where Y is the dependent variable, β0 is the intercept, and β1 and β2 are the coefficient of the two features x1 and x2, respectively.

With the model trained, we can obtain the intercept as well as the coefficients of the features:

Getting the Intercept and Coefficients

After building a model based on the training set, we can use the model to determine both the *intercept* and the *coefficients*.

In [ ]:

```
print("Model Intercept   :",model.intercept_)
print("Model Coefficient :" ,model.coef_)
```

```
Model Intercept   : 0.567856466318923
Model Coefficient : [ 0.43191852 -0.03240486]
```

We can also use the esitmator's `predict()` method to use the model to predict the house price given a specific value for `MedInc` (2.5) and `AvgRooms` (2) ...

In [ ]:

```
print(model.predict([[2.5,2]]))
```

```
[1.58284304]
```

```
C:\Users\shrih\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\base.
py:441: UserWarning: X does not have valid feature names, but LinearRegression was fi
tted with feature names
  warnings.warn(
```

Model Equation for Multiple Linear Regression

The equation can be verified using the predicted value by using the formula that was given earlier:

The formula for multiple regression is as follows: Y = β0 + β1x1 +β2x2

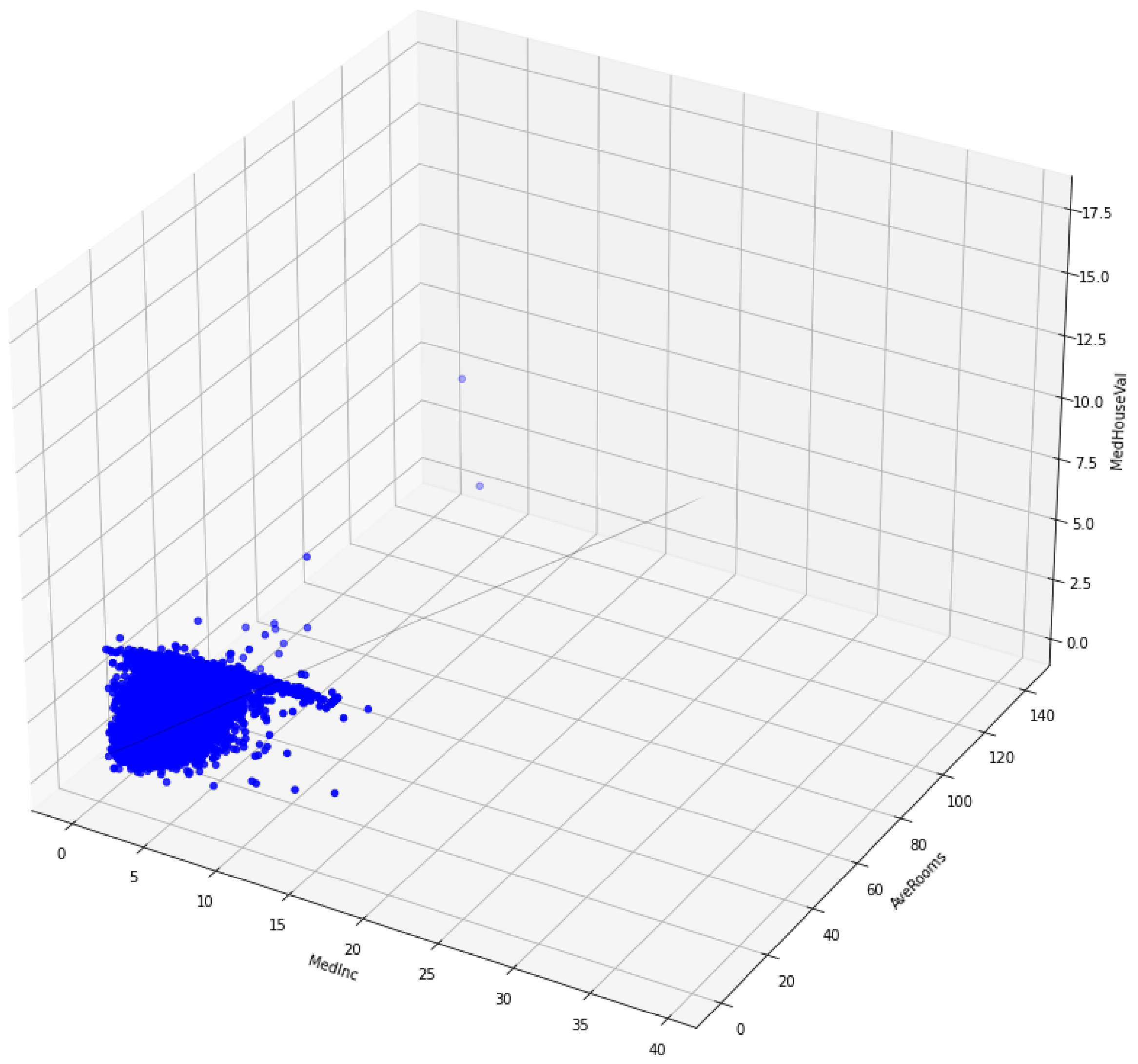
where Y is the dependent variable, β0 is the intercept, and β1 and β2 are the coefficient of the two features x1 and x2, respectively.

Y = 0.567856466318923 + (2.5) * 0.43191852 +(2) (-0.03240486) = 1.58284304

In this way , using equation we can verify these values.

Plotting the 3D Hyperplane for Multiple linear regression

In [ ]:
```python
fig = plt.figure(figsize=(18,15))
ax = fig.add_subplot(111, projection='3d')

ax.scatter(x['MedInc'],
           x['AveRooms'],
           Y,
           c='b')

ax.set_xlabel("MedInc")
ax.set_ylabel("AveRooms")
ax.set_zlabel("MedHouseVal")

#---create a meshgrid of all the values for MedInc and AveRooms---
x_surf = np.arange(0, 40, 1)   #---for MedInc---
y_surf = np.arange(0, 10, 1)   #---for AveRooms---
x_surf, y_surf = np.meshgrid(x_surf, y_surf)

from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x, Y)

#---calculate z(MEDC) based on the model---
z = lambda x,y: (model.intercept_ + model.coef_[0] * x + model.coef_[1] * y)

ax.plot_surface(x_surf, y_surf, z(x_surf,y_surf),
                rstride=1,
                cstride=1,
                color='None',
                alpha = 0.4)

plt.show()
```



Polynomial Regression

As far you saw how to apply linear regression to predict the MedHouseVal in California housing dataset. While the result is somewhat acceptable, it is not very accurate.

This is because sometimes a linear regression line might not be the best solution to capture the relationships between the features and label accurately. In some cases, a curved line might do better.

Now let's try to apply it to the California housing dataset and see if we can improve the model.

In [ ]:
```
california_housing_df = pd.DataFrame(california_housing.data, columns=california_housi
california_housing_df['MedHouseVal'] = california_housing.target

x = pd.DataFrame(np.c_[california_housing_df['MedInc'], california_housing_df['AveRoom
columns = ['MedInc','AveRooms'])
Y = california_housing_df['MedHouseVal']
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, Y_train, Y_test = train_test_split(x, Y, test_size = 0.3,
                                                    random_state=5)
```

Using this PolynomialFeatures object, you can generate a new feature matrix consisting of all polynomial combinations of the features with a degree of less than or equal to the specified degree:

```
In [ ]: #---use a polynomial function of degree 2---
degree = 2
polynomial_features= PolynomialFeatures(degree = degree)
x_train_poly = polynomial_features.fit_transform(x_train)
```

When using a polynomial function of degree 2 on two independent variables x1 and x2, the formula becomes:

equation Y =β0 + β1x1 +β2x2 + β3x1^2+ β4x1x2 +β5x2^2

where Y is the dependent variable, β0 is the intercept, and β1, β2, β3, and β4 are the coefficients of the various combinations of the two features x1 and x2, respectively.

You can verify this by printing out the feature names:

```
In [ ]: #---print out the formula---
print(polynomial_features.get_feature_names(['x','y']))
```

```
['1', 'x', 'y', 'x^2', 'x y', 'y^2']
C:\Users\shrih\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\utils
\deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feat
ure_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_nam
es_out instead.
  warnings.warn(msg, category=FutureWarning)
```

You can then train your model using the LinearRegression class

```
In [ ]: model = LinearRegression()
model.fit(x_train_poly, Y_train)
x_test_poly = polynomial_features.fit_transform(x_test)
print('R-Squared: %.4f' % model.score(x_test_poly,Y_test))
```

```
R-Squared: 0.5082
```

You can also print the intercept and coefficients

```
In [ ]: print(model.intercept_)
print(model.coef_)
```

```
0.6262082852484339
[ 0.          0.56791614 -0.11818639 -0.01185017  0.00289779  0.00097166]
```

Model Equation for Polynomial Linear Regression

With these values model equation becomes:

Y =β0 + β1x1 +β2x2 + β3x1^2+ β4x1x2 +β5x2^2

where Y is the dependent variable, β0 is the intercept, and β1, β2, β3, and β4 are the coefficients of the various combinations of the two features x1 and x2, respectively.

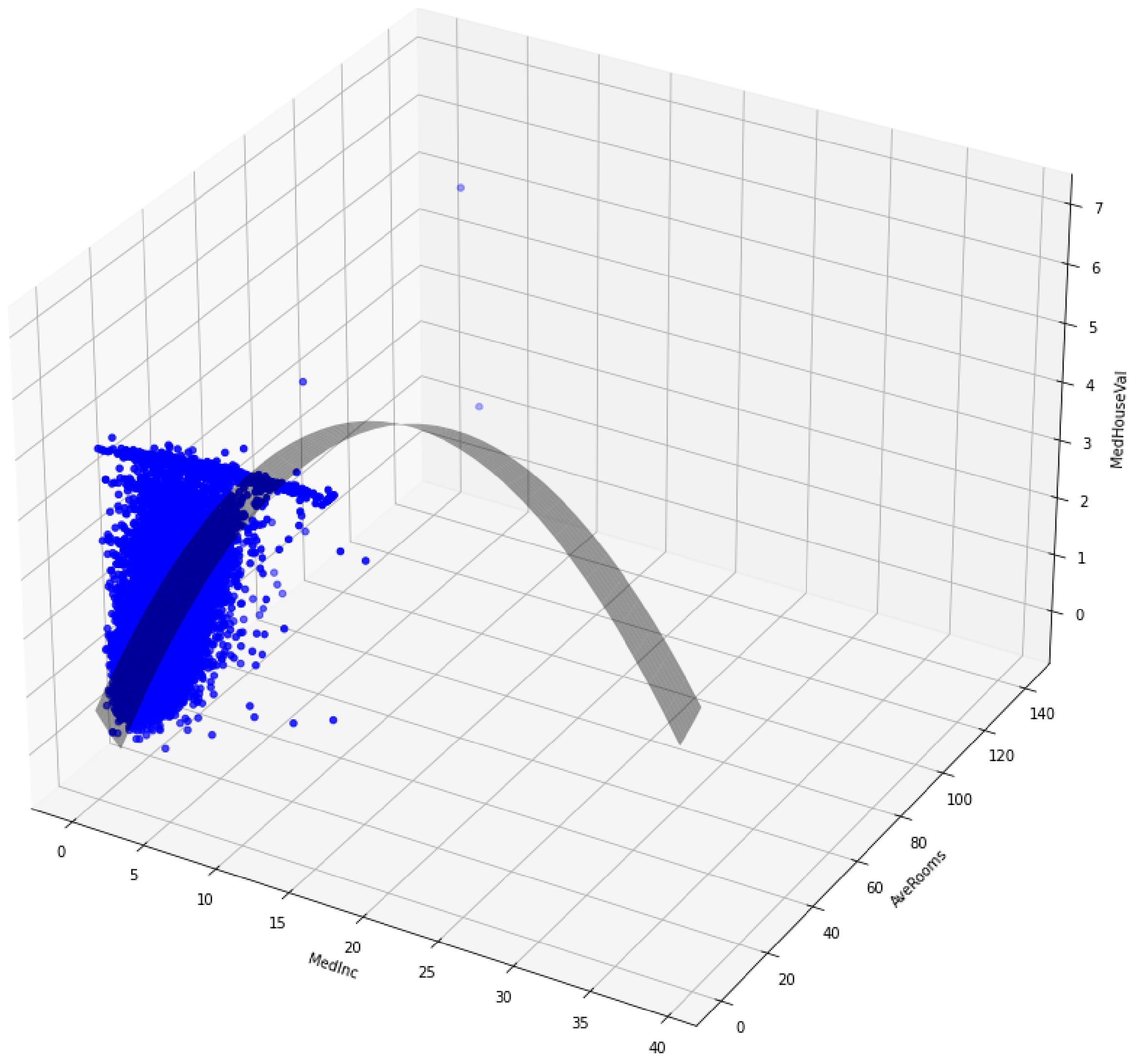
Y = 0.6262082852484339 + (0.56791614) x1 +(-0.11818639)x2+(-0.01185017)x1^2+(0.00289779)x1x2 +(0.00097166)x2^2

Plotting the 3D Hyperplane for polynomial regression

In [ ]:
```
fig = plt.figure(figsize=(18,15))
ax = fig.add_subplot(111, projection='3d')

ax.scatter(x['MedInc'],
           x['AveRooms'],
           Y,
           c='b')

ax.set_xlabel("MedInc")
ax.set_ylabel("AveRooms")
ax.set_zlabel("MedHouseVal")

#---create a meshgrid of all the values for LSTAT and RM---
x_surf = np.arange(0, 40, 1)   #---for LSTAT---
y_surf = np.arange(0, 10, 1)   #---for RM---
x_surf, y_surf = np.meshgrid(x_surf, y_surf)

#---use a polynomial function of degree 2---
degree = 2
polynomial_features= PolynomialFeatures(degree = degree)
x_poly = polynomial_features.fit_transform(x)
print(polynomial_features.get_feature_names(['x','y']))

#---apply linear regression---
model = LinearRegression()
model.fit(x_poly, Y)

#---calculate z(MEDC) based on the model---
z = lambda x,y: (model.intercept_ +
                (model.coef_[1] * x) +
                (model.coef_[2] * y) +
                (model.coef_[3] * x**2) +
                (model.coef_[4] * x*y) +
                (model.coef_[5] * y**2))

ax.plot_surface(x_surf, y_surf, z(x_surf,y_surf),
                rstride=1,
                cstride=1,
                color='None',
                alpha = 0.4)

plt.show()
```

```
['1', 'x', 'y', 'x^2', 'x y', 'y^2']
```

```
C:\Users\shrih\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\utils
\deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feat
ure_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_nam
es_out instead.
  warnings.warn(msg, category=FutureWarning)
```



Summary and Conclusion

Model Equation:

1. Model Equation for Multiple Linear Regression

```
 The equation can be verified using the predicted value by using the
formula that was given earlier:
 The formula for multiple regression is as follows:
 Y = β0 + β1x1 +β2x2
 where Y is the dependent variable, β0 is the intercept, and β1 and β2
are the coefficient of the two features x1 and x2, respectively.
 Y = 0.567856466318923 + (2.5) * 0.43191852 +(2) (-0.03240486) =
1.58284304
 In this way , using equation we can verify these values.
```

2. Model Equation for Polynominal Linear Regression

```
 With these values model equation becomes:
 Y =β0 + β1x1 +β2x2 + β3x1^2+ β4x1x2 +β5x2^2
 where Y is the dependent variable, β0 is the intercept, and β1, β2, β3,
and β4 are the coefficients of the various combinations of the two
features x1 and x2, respectively.
 Y = 0.6262082852484339 + (0.56791614) x1 +(-0.11818639)x2+
(-0.01185017)x1^2+(0.00289779)x1x2 +(0.00097166)x2^2
```

R Square Value: The R-Squared method lets you know how close the test data fits the regression line. A value of 1.0 means a perfect fit. So, you aim for a value of R-Squared that is close to 1

1. R Square Value for Multiple Linear Regression : 0.4924
2. R Square Value for Polynominal Linear Regression: 0.5082

As there is improvement in R-Square value, so we are able to improve model with polynomial regression